공학석사학위논문

McDRAM: DRAM에서의 낮은 대기 시간 및 에너지 효율적인 행렬 연산

McDRAM: Low Latency and Energy-Efficient Matrix Computation in DRAM

2018년 2월

서울대학교 대학원

컴퓨터공학부

신 현 승

McDRAM: DRAM에서의 낮은 대기 시간 및 에너지 효율적인 행렬 연산

McDRAM: Low Latency and Energy-Efficient Matrix Computation in DRAM

지도교수 유 승 주

이 논문을 공학석사 학위논문으로 제출함

2017년 10월

서울대학교 대학원

컴퓨터공학부

신 현 승

신현승의 공학석사 학위논문을 인준함

2017년 12월

위 원 장 _____

부위원장 _____

위 원 _____

Abstract

McDRAM: Low Latency and Energy-Efficient Matrix Computation in DRAM

Shin Hyunsung

Department of Computer Science & Engineering

The Graduate School

Seoul National University

Neural networks are characterized by massively parallel computation and high memory bandwidth. In particular, memory bandwidth severely limits performance and increases power consumption. In order to overcome memory bottleneck of neural network applications, we propose a novel memory architecture called McDRAM where DRAM dies are equipped with a large number of multiplier-accumulator (MAC) units to perform neural networks internally. Each bank of DRAM memory has multiple MACs as much as the size of memory pre-fetch data, thereby fully utilizing internal bandwidth of DRAM which far larger than external memory bandwidth. McDRAM broadcast data efficiently to all bank

without any modifications of DRAM data bus, and it performs MAC operations in the all banks with a single DRAM command. McDRAM is implemented based on the state-of-the-art commercial memory architecture, HBM2, and it equips thousands of MACs (up to 6,144 in HBM2) in a single DRAM package. According to our experiments with in-house memory models based on commercial JEDEC HBM2 simulator, McDRAM achieves 18.68x TOPS/W performance compared to the state-of-the-art hardware accelerator (Google TPU) in LSTM.

Key words: Neural Network, DRAM, RNN, LSTM, MLP, MAC, HBM2

Student Number : 2016-21217

Contents

1. Introduction

As neural networks (NNs) are widely applied to various applications from servers to mobile devices, the necessity of dedicated chip design is growing due to performance and energy efficiency [1–4]. In server applications, the neural networks are mostly applied to user-interactive scenarios (e.g., recognizing voices, predicting user's next clicks, and etc.) which favors multi-layer perceptron (MLP) and recurrent neural network (RNN) to meet tight latency requirements. The majority (about 90% [5]) of neural networks deployed in the server are MLP and RNN, thus performing MLP and RNN efficiently is key to optimize NN in server applications. The LSTM (Long Short-Term Memory) used in the experiment is a kind of RNN.

Early-stage NN accelerators suffered from large performance and energy overhead due to high off-chip memory traffics [2]. In order to mitigate this memory bottleneck, recent hardware accelerators adopt enough on-chip memory to provide high bandwidth for accessing reuse-friendly data [6]. Especially, this approach effectively reduces the off-chip memory accesses for

convolutional neural networks (CNNs). However, this approach is not adequate for providing enough bandwidth in MLPs and RNNs, because in these networks (1) weights tend to be used only once, and (2) it is difficult to scale up on-chip memory (e.g., SRAM, eDRAM, and etc.) enough to store data.

The majority of computations in MLPs and RNNs are matrix-vector (MV) multiplication which is characterized by low data reuse (i.e., each MV multiplication uses the same weight only once) compared with convolution operation (i.e., each convolution operation reuses the same weight multiple times). What is worse, due to the large model size (usually hundreds of megabytes) of MLPs and RNNs on servers, on-chip memory cannot afford to store every weight. Thus, the performance of MLP and RNN is often determined by the available off-chip memory bandwidth [5].

For this reason, in order to realize fast/energy-efficient NN, many companies adopted High Bandwidth Memory (HBM) while designing NN specialized hardware accelerator. However, current NN accelerators utilizing HBM are limited in server applications due to two reasons. First, it requires high design effort because several I/O interfaces complexly affects the cost, and large energy

consumption due to I/O interface limits its usage to meet the power budget.

In order to avoid overhead due to I/O interface and overcome memory bottleneck problem of NN applications, we propose a new memory architecture called McDRAM which equips with multiplier-accumulator (MAC) units. McDRAM efficiently utilize internal DRAM bandwidth to perform NNs which require large amount of data with low power consumption. Our contributions are as follows.

- We present a novel DRAM microarchitecture, called McDRAM, where a DRAM die is equipped with thousands of MAC units.
- McDRAM equips MAC units in Column Decoder of DRAM banks, thereby minimizing the energy consumption for data transfer. Furthermore, it performs multiple batch operations with only one read operation of weights which can be able to increase performance in proportion to the number of banks of DRAM.
- According to our experiments, McDRAM achieves 18.68x and 23.84x better TOPS/W performance compared to TPU

and high energy efficient GPU (NVIDIA P4) in LSTM.

This paper is organized as follows. Section 2 gives background and motivation. Section 3 describes our proposed architecture, McDRAM. Section 4 describes McDRAM scheduling. Section 5 explains our evaluation methodology. Section 6 reports experimental results. Section 7 reviews related work. Section 8 summarizes the paper.

2. Background and Motivation

2.1 Memory Bandwidth in NN

The MLP requires large memory bandwidth due to two reasons. First, the weights are not reused in MLP. A new set of weights should be fetched from the main memory to compute each output activation, thereby degrading performance and power efficiency. Furthermore, since shallow and wide MLP is preferred in server applications to reduce latency, such a wide MLP imposes a significant capacity pressure on local memory on chip. In this case, the performance of wide MLPs is determined by off-chip memory bandwidth. Note that similar problem is observed in RNN which consists of large a number of MV multiplications.

Recent NNs continue to demand the processing of larger data, and in this situation the memory bandwidth will continue to be the biggest obstacle to NNs in the future. With the development of artificial intelligence, new networks such as Google's Memory Network are emerging, but we think that Its performance will also be limited by memory bandwidth. However, it is very difficult to mount a very large SRAM in Accelerator and it is also very difficult

to use high bandwidth memory such as HBM2 to increase DRAM bandwidth.

Accelerators can use more memory or higher bandwidth memory to increase memory bandwidth, but in this case high power consumption is caused by the interface. This is a difficult decision because performance is related to memory bandwidth, but it is a trade-off relationship where accelerators want to increase performance but higher power consumption. In NN, there is a desperate need for a method which requires high memory bandwidth and low power consumption.

2.2 DRAM Architecture

Since McDRAM adds MACs and logics to the existing DRAM structure, we introduce the components of DRAM in this section.

2.2.1 Channels and Banks. In DRAM, there exists a channel and a bank in order to independently process data in parallel. There are several banks on a DRAM die which share the same signal and data-path. Bank block operates as interleaving while performing operations, and thus, it transfers data with high bandwidth. In a

DRAM composed of several dies and/or several packages, a channel transfers data at a high bandwidth in the same manner. Without any design changes, the bank shares the data, so it can broadcast the data in one data-path at a time. Moreover, with the small design changes, data can be easily broadcasted to multiple channels with a single DRAM command when connecting multiple data-path channels.

2.2.2 I/O Sense Amplifiers. Since DRAM cells store data by charging tiny electric charge, several amplifications are required to fetch the data. The first amplifier in DRAM, called Bit Line amplifier, uses differential signals which are converted to CMOS level data after passing through the IOSA (I/O sense amplifier) in the Column Decoder. This data can be processed for logic operations with the least amount of power consumption.

2.2.3 Data Pre-Fetch Size. Since DRAM cell consists of one transistor and one capacitor, the speed cannot be increased fundamentally. For this reason, DRAM has been speeding up by fetching multiple data at the same time and outputting data serially.

Table 1 shows the pre-fetch sizes and the maximum speeds of several types of DRAM.

Table 1: Pre-Fetch Size in DRAM types

DRAM	Pre-Fetch Size	Maximum Speed
SDR	1	133Mbps
DDR	2	400Mbps
DDR2	4	1066Mbps
DDR3	8	2133Mbps
DDR4	8	3200Mbps

As shown in Table 1, most recent DRAMs use 8 pre-fetch sizes for high speed. The 8 pre-fetch size is very effective to use 8bit MAC operation.

2.2.4 DQ Blocks. Each bank of DRAM is composed of several DQ blocks. Each DQ block is connected to one data bus pin while performing read/write operations, and it exchanges data by pre-fetch size. DQ block consists of several memory cells MAT, and each DQ block contains one IOSA. For example, DDR4 x8 products consist of 8 data pins, and thus, 8 DQ blocks exist in each bank. In the same manner, LPDDR4 x32 products contains 32 data pins, as well as, 32 DQ blocks in each bank. One bank is enabled by one active command. When active in LPDDR4 x32, 32ea of WLs (Word

Lines) are enabled and ready to read/write data.

2.2.5 SerDes(Serialization & Deserialization). When the Read command is issued, the inside of the DRAM cell operates in parallel. However, in order to send data at high speed to the outside, data should be serialized and transmitted. On the contrary, when the Write command is issued, the serial data should be parallelized. Such a SerDes operation consumes large energy in DRAM

2.2.6 DQ Termination. To transfer data at high speeds and eliminate signal reflection in DRAM interface, DRAM utilizes termination. However, termination consumes a lot of interface energy. Current DRAMs require more than 50% energy to interface with external interface.

2.3 Motivation

We perform the multiplication & addition operation with the lowest power by deploying the MAC logics into the existing DRAM Column Decoder. As shown in Figure 1, amplified signals from DRAM cell, GIO and GIOB, are converted into a single-ended CMOS

signal through the IOSA. While performing MV operation, data path connecting IOSA and DQ_OUT is disabled to reduce unnecessary energy. We utilized data from IOSA as an input data of MAC unit (grey block in Figure 1) to compute MV operation.

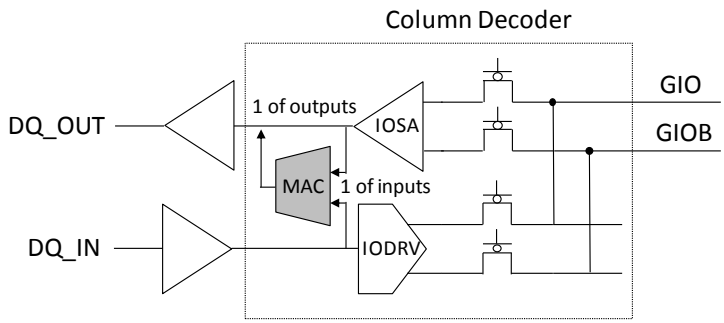


Figure 1: MAC in DRAM Column Decoder

In McDRAM, we utilize conventional DRAM command and blocks for normal operations, and furthermore, exploiting current DRAM commands, we implement commands for MAC operation. For example, in order to write data into DRAM, we execute basic Write command and store in the DRAM cell through the IODRV unit existing in current DRAM structure. While performing MAC operation, the source of each input of MAC pushes data continuously. These operations are performed simultaneously like data read operation of conventional DRAM. Thus, exploiting Read

command, McDRAM performs MAC operations without degrading performance.

Table 2: Current Consumption in 3D TSV DRAM

Unit	Current Consumption
Local Sense Amplifier	1
GIO	10
TSV Driver	200
External I/O Driver	600

Table 2 shows the normalized current consumption of each amplifiers connecting DRAM cell and the external I/O. In order to transfer data from DRAM cell to outside, data pass through local sense amplifier, GIO, TSV driver and external I/O driver in sequence. Since bandwidth shrinks, data originally processed in parallel should be serialized as progress to the next stage. More worse is that current consumption increases as data transferred to the next level. To pass through GIO unit, 10 times larger current is consumed compared to local sense amplifier. Data passing through GIO unit consumes 200 times larger current to pass through TSV driver. Finally, in order to transfer data outside, external I/O driver consumes 600 times larger current.

Memory (usually DRAM) and computation blocks (usually

containing SRAM) of conventional NN accelerators are separated. Thus, conventional NN accelerators read data through external I/O driver of DRAM and feed them into computation blocks. However, McDRAM poses computation units in the column decoder of DRAM, thereby transferring data only up IOSA to perform computation. Compared to the conventional NN accelerators which consumes 600 times larger current by accessing external I/O driver, McDRAM consumes less current by accessing IOSA. Moreover, McDRAM does not serialize data which is originally stored in parallel at each bank, because it does not transfer data up to external I/O driver via TSV driver. By doing so, proposed McDRAM mitigates memory bottleneck problem of NN applications and reduces energy consumption.

3. McDRAM Architecture

In order to perform the MV operations, McDRAM uses the broadcast data and the data stored in the DRAM cell as shown in Figure 2 below. Depending on the type and size of NN, the data to be sent to the broadcast depends on the selection. We use the activation data with broadcast data on CNNs. And we use the weight as the broadcast data in such as MLPs and RNNs. For large-sized Matrix operations such as MLPs and RNNs, different input batches are stored in each independent bank. In Figure 2, the data for Batch 1 is stored in Bank 1 and the data for Batch 2 is stored in Bank 2. In this way, it can store multiple batches as many as the total number of banks. With sufficient batch size inside the DRAM, the large-sized weight data in Figure 2 is executed only once.

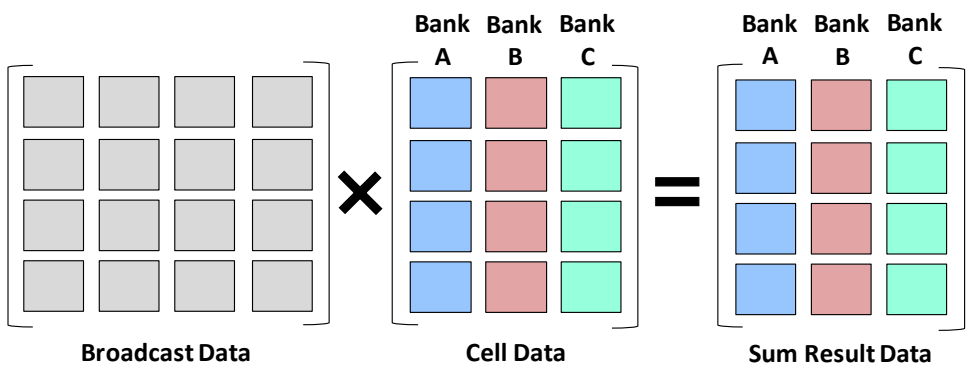


Figure 2: MV (Matrix Vector) Multiplication

Depending on the predefined batch size, the MV is used only once, no recycling is required, and there is no need to store it separately inside the DRAM. It means that the operation with a very large weight can be performed very well. Since the batch size that operates in McDRDAM is set in advance, it can complete the MV operation with minimum time if software operates only in the batch size. If the batch size is greater than the number of banks supported, it must access more than one weight data. The number of banks supported by DRAM in hardware is determined by the total number of banks of one channel. Since McDRAM sends data to several channels in common, it is possible to store batches as many times as the number of banks multiplied by the number of channels.

3.1 Overview

HBM2 memory has 8 independent channels, each channel has independent command signal and data bus. If the weight data is stored in a place other than the current McDRAM, McDRAM can use only one channel interface to input the weight externally, and MAC operation can be performed simultaneously on 8 channels. If the weight data is stored in an external nonvolatile storage space or

stored in another memory, McDRAM can use one channel interface and the MAC operation can be performed simultaneously on all 8 channels. However, in order to minimize the power consumption by the interface when there is no any external memory, McDRAM uses 2 broadcast channels in 8 HBM2 channels as shown in Fig 3.

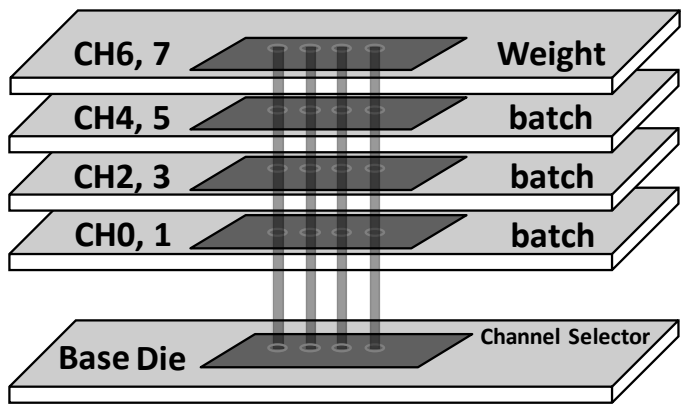


Figure 3: HBM2 based McDRAM Architecture

First, the general memory write method is used to store the weight of NN in CH6 and CH7 before MV operation, and the batch data to be executed should be distributed into CH0 to CH5. Depending on the design, McDRAM has only one channel that interfaces with the external I/O. The channel selector of the base die (logic die) in the figure above allows host to select the corresponding channel. McDRAM does not have any memory interface after the weight and batch data required for MV operation

are written. After all NNs are done, the results can be fetched through I/O lastly.

Our goal is to maximally exploit the high degree of parallelism in matrix multiplication operations based on the inherent parallel structures in DRAM architectures, while minimizing the additional hardware cost for practicality. We have placed a MAC for each DQ block in a DRAM column decoder in order to utilize the existing DRAM's data-path as much as possible without any change. Figure 4 shows the new MAC, Adder Tree, and MUX in the Column Decoder in McDRAM.

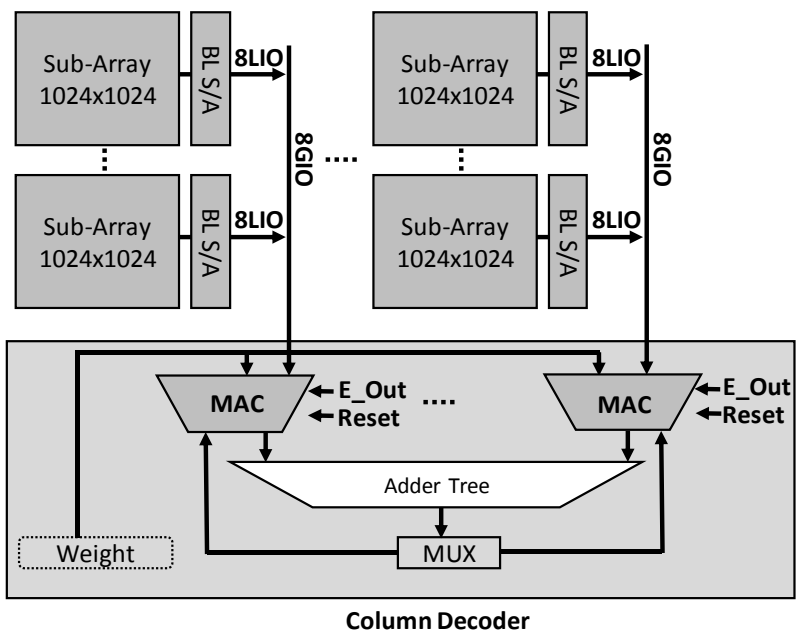


Figure 4: HBM2 based McDRAM Architecture

We will describe in Section 3.2 how the weight data is broadcasted through conventional DRAM data-path. In Section 3.3, we describe how to fetch data from a DRAM cell where batch data is stored. In Section 3.4, we will discuss how to perform MAC operations on the two preceding data. We will then discuss how to process the MAC result in Section 3.5 and the Activation Function used in NN in Section 3.6.

3.2 Broadcast Data-Path

HBM2 operates at 2Gbps per pin and has a total of 1024 I/O buses. So the total bandwidth is 256GBps. HBM2 has 8 channels in total, and one channel has 128 I/O buses. HBM2 operates by dividing into Pseudo-0 and Pseudo-1 to effectively interleave data with higher bandwidth, and each Pseudo has 64 bit I/O buses. Since HBM2 operates as BL4, one pseudo interface uses 128 x 4 bits data which means it is operating by 32bytes. Figure 5 shows that CH6 or CH7 in McDRAM broadcast 32 bytes of data from CH0 to CH6, respectively, through the Channel Selector.

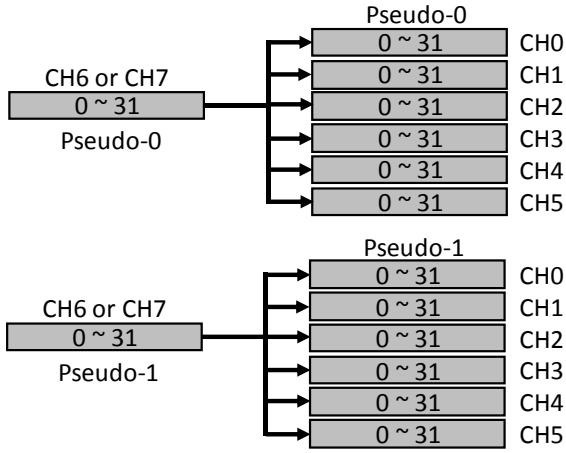


Figure 5: Broadcast Data in McDRAM

The data sent from CH6 and CH7 is transmitted to the base die, which is the bottom die, through the TSV line. When the broadcast signal is enabled by the Channel Selector, this data is simultaneously transmitted to the six channels. At this time, data is also transmitted through the TSV line. Basically there is latency because TSV line goes through the base die. However, since the throughput for MAC operation is the same as the conventional one, the performance is not affected.

3.3 DRAM Cell Data Fetch

The DRAM consists of a MAT with 1024x1024 or similar size MAT. Multiple MATs are connected to the IOSA in the Column Decoder via a common bus called GIO as shown in Figure 4. In HBM2, one pseudo block outputs 128 x 4 bits of data at a time, so it has 512 IOSAs in one bank. McDRAM assumes that the size of the MAT is 1024x1024, and when one active command is issued, a total of 64 WL (Word Line) is enabled in one bank of HBM2 which means two pseudo blocks, so 8KBytes of data can be used as gapless without any pre-charge command.

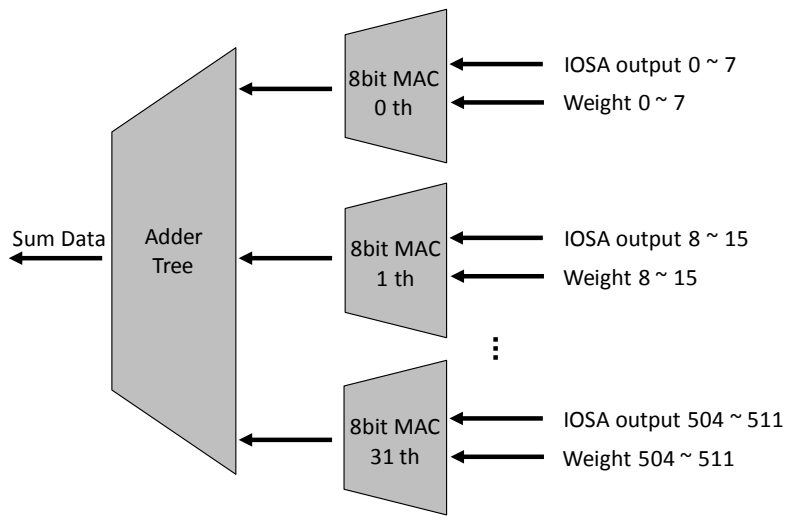


Figure 6: 32 MAC Units of One Pseudo Block

3.4 Matrix-Vector Multiplication

McDRAM has a total of 96 banks of 6 channels. As mentioned earlier, each bank operates independently and stores different batch data. Figure 7 below shows that data is delivered to 96 banks with weight data when operating as a pseudo. In a single pseudo block, 32 bytes of data are transmitted in common to 96 banks of inputs, and each bank has 32 MAC units each. Therefore, one pseudo block has a total of 96 x 32 = 3,072 MACs and one channel consists of two Psedo blocks. Therefore, the total MAC of the McDRAM has a total of 6,144 MACs.

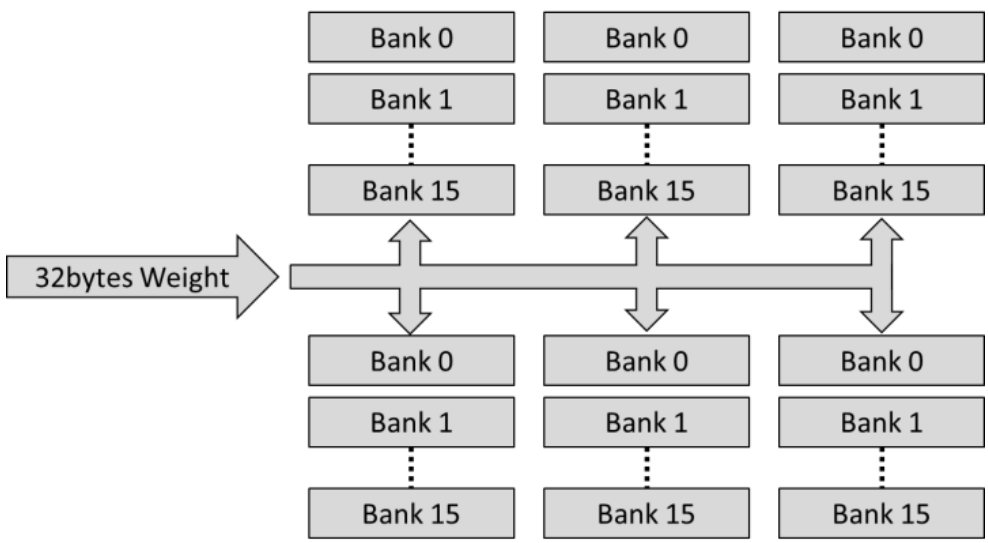


Figure 7: 96 Banks of McDRAM

In order to operate the MAC operation continuously, the

input data to be entered to the MAC unit must be fetched from DRAM cell continuously. After WL is enabled, the data output by the CSL is determined by the tCCD parameter of the DRAM Specification. tCCD means Column to Column Delay of DRAM core parameter. Since HBM2 uses BL4 and uses DDR(Double Data Rate), tCCD becomes 2 clocks. And HBM2 uses 1Ghz operating frequency and data frequency at 2Gbps, so gapless operations can be operated by 2ns period.

Since HBM2 can fetch data in 2ns unit, we designed 500Mhz MAC. Thus, McDRAM has 6,144 8bit MAC Units with an operating frequency of 500Hhz. By the way, it is very difficult to insert a MAC of 500Mhz when mounting a MAC in a different type of memory other than HBM2. In case of HBM2 or GDDR5, tCCD is 2ns because it is designed for high bandwidth. However, memory used for other applications such as DDR4 or LPDDR4 cannot accommodate 500Mhz MAC because tCCD is larger than this value.

3.5 Sum Aggregation

There are 64 MAC Units in one bank, and the results of the MAC Unit are input to the Adder Tree. The Adder Tree has 32 +

16 + 8 + 4 + 2 = 62 Adders, and the final value of the Adder Tree is the result of one batch when the Matrix operation is actually performed. Figure 8 shows the circuit that processes the result after Matrix operation. This result value can be transmitted by the MUX in two data-paths according to the command. This option depends on the type of NN and how the results of the Matrix are handled.

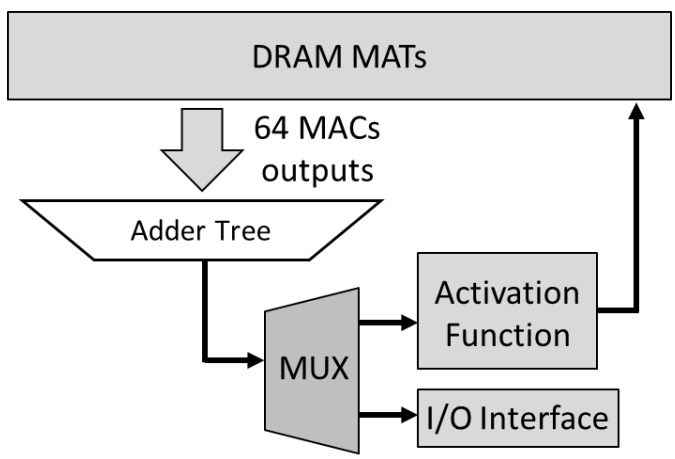


Figure 8: Matrix Result Data-path

When McDRAM receives the first option as a command, the result of the Matrix operation is directly entered into the input of the Activation Function unit. After this unit finishes the operation, the result value returns back to the DRAM MATs. This method reduces power consumption because McDRAM does not have to

send the result to the outside. And the performance is higher because the result of the batch needed for the next calculation is restored. Currently, McDRAM can use 8Kbytes of batch data. With this method, however, McDRAM can only use half of this value, so the size of the batch that can be computed is halved.

The second option has the advantage of outputting the result of the Matrix to the outside without using it in the next stage. And because this method can use 8Kbytes of weight data, it can perform bigger matrix than the first method. One special design is that since 16 banks in a DRAM die share a 128-bit data bus, each bank is assigned a corresponding DQ bus number according to its bank number. For example, Bank 0 uses 0 to 7 DQ buses among 128 data buses. And since BL (Burst Length) is 4, McDRAM can transmit 4 data with one command. Therefore, 4 results can be obtained by inputting command once in 4 times. McDRAM can write multiple batch data using a single data-bus in the opposite direction.

3.6 Activation Function

As shown in Fig. 8, McDRAM has an Activation Function unit in the Column Decoder. Since there is only one activation function

unit per DRAM bank, the overhead is not large. In addition, McDRAM has logic for maxpooling in the base die, and Maxpooling is currently operating in units of 32bytes.

4. McDRAM Scheduling

Since all operations are operated by the Memory Command, the performance of the McDRAM depends on the scheduling of the input command. In order to maximize the memory bandwidth, the memory controller operates as bank interleaving and is designed to fully utilize the page opened by one activation command. we modeled the scheduling in the same way. We used the open page as much as possible to broadcast the weight data, thereby increasing the effective bandwidth and consequently improving the overall performance.

We simulated the memory controller currently operating in the system. The existing memory controller has a command buffer inside it to check the history of commands list and to open or close the page. In general, the memory controller cannot use memory bandwidth ideally because the memory controller does not know the characteristics of the system workload. In addition, since existing memory controller process instructions and data at the same time, performance can be reduced by instruction fetch operation even when processing high bandwidth of stream data.

However, when using McDRAM for processing purposes,

McDRAM can access memory with ideal memory bandwidth by issuing only location and size of the matrix into McDRAM in the host because it knows in advance before processing the memory sequences to be processed. If the batch size is less than 8Kbytes, McDRAM can access the memory with high bandwidth because it only accesses the activated page to fetch batch data stored in each bank without any other page activation. If the batch data exceeds 8Kbytes, McDRAM will process the data with minimum page activation. For example, when processing batch data with a size of 16Kbytes, the first 8Kbytes is taken first, then all the data is extracted from one page, and the remaining 8Kbytes are extracted from the next page. However, performance degradation occurs because all the banks are precharged and all the banks are activated again. On the other hand, since weight data is processed by interleaving on several banks, it is less restrictive than activating a page when processing batch data.

4.1 Input Data Load

McDRAM can process all data processing only in 32bytes. So input data must be split into 32bytes in one pseudo. Especially,

when inputting the first image of CNN, it is sometimes inefficient to process the size of 32 bytes. Figure 9 below shows how to process input images in a ALEXNET network. Since a pixel in image is very small in size of 3, we loaded the image in 1x11 size of convolution. Since the input image is entered as 1x11, the actual convolution operation is changed from the existing 11x11 to 11x1. In addition, since the stride is 4 in the y-axis direction, 4 pixels cannot be extracted in the current data type. Therefore, when the input image is loaded, the input image is used redundantly. However, most of the processing after input layer can handle the data very efficiently with 32 bytes.

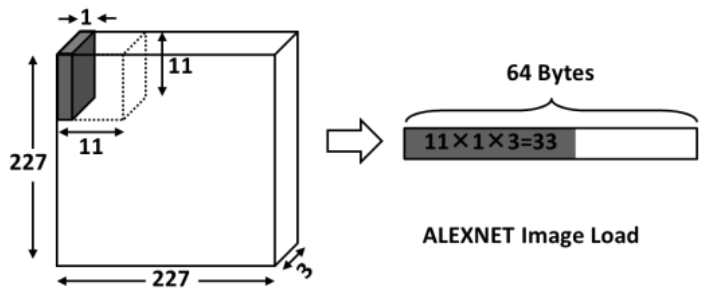


Figure 9: Processing of ALEXNET input image

4.2 Configuration Output Data Deployment

The final result of the Matrix in McDRAM can handle a variety of data using the two options described in 3.5 Sum

Aggregation. Using the first option to put the final result back on the same page allows only 4Kbytes per page to be used as mentioned above, so McDRAM can use two 4Kbytes of space to perform the entire network . If McDRAM use this method, not only the batch data but also the intermediate data is rewritten, so only the final result is stored. It can be used efficiently in a network that has many middle layers such as MLP and RNN. If McDRAM use this option to store the final result on a different page, you will have to perform a per-matrix operation to close the page where the current batch is stored and to activate another page. However, since most of the matrices are large in size, this single operation does not cause much performance degradation.

If McDRAM use the second option described in Section 3.5, McDRAM can output the result of the matrix operation to external I / O each time. However, if a network that reuses intermediate layer results such as MLP and RNN is used, it is necessary to write a batch beforehand when computing the next layer's matrix. When writing batch data, as mentioned above, each bank has corresponding DQ bus so that all batch data can be written at once by data bus corresponding to one channel interface.

4.3 Broadcast and MAC Operation Scheduling

McDRAM uses the existing memory interface and all operations are performed as memory commands. These McDRAM instructions are compatible with conventional Memory instruction interfaces. The biggest feature of the commands of McDRAM is that the MAC operation is run across all the banks. Just as if you give Refresh command in DRAM, MAC command also allows all banks to work at the same time as all banks do work at the same time.

Except external interface power, DRAM consumes the most amount of current when Refresh command comes in. Since Refresh activates the page in all banks, a large amount of current is consumed instantaneously. When activating in one bank, all BL S/A in MAT operates, so there is a lot of current consumption. Therefore, the refresh command of the DRAM activates the page by dispersing the banks during the tRFC time. When the page is activated, McDRAM processes the same as the existing DRAM command. All channels containing batch data consume a lot of current only when McDRAM first activates the data. And since the current consumed when using the I/O interface is much larger than the current consumed when performing the MAC operation, it is not

influenced by the power budget in McDRAM.

Table 3 shows the new commands added in McDRAM. The BRO command fetches the data stored in the channel according to the number of the selected channel and broadcasts the data through the channel selector of the base die. The MAC command fetches the data stored in the current MAT in the 96 banks as described above, and inputs them into the MAC input. The MRST instruction is the command to reset the data currently stored in each MAC, and must be executed first when performing Matrix operation. The SUM command sends the final result values stored in the current MAC to the Adder Tree so that all 64 data are summed up, and the result is stored in the column decoder area. Finally, the MWRT instruction is used to save the data stored in the current column decoder to MAT again.

Table 3: McDRAM Command

Command	Description
BRO	Data Broadcast to All 96 Banks
MAC	MAC Operation in 96 Banks
MRST	Reset the value of MAC
SUM	Read Result
MWRT	Write Result Data

Figure 10 below shows the timing diagram of McDRAM using pseudo mode. HBM2 operates at 1GHz and Tn operates at 1ns. Since both broadcast and MAC operations operate in pseudo mode, pseudo 0 and pseudo 1 are input alternately. McDRAM receives the weight data with the BRO command first, and then fetches the batch data from the MAT with the MAC command. In the figure, the BRO command and the MAC command are drawn at intervals of 1 ns. In practice, however, this command cannot be entered because the BRO command via TSV has a longer latency than the MAC command. However, even if the BRO command has a long latency, the actual bandwidth to be processed is the same as the MAC command. Therefore, Internally the BRO command can be executed in advance or the MAC command can be delayed so all operations can be operated as shown in Figure 10.

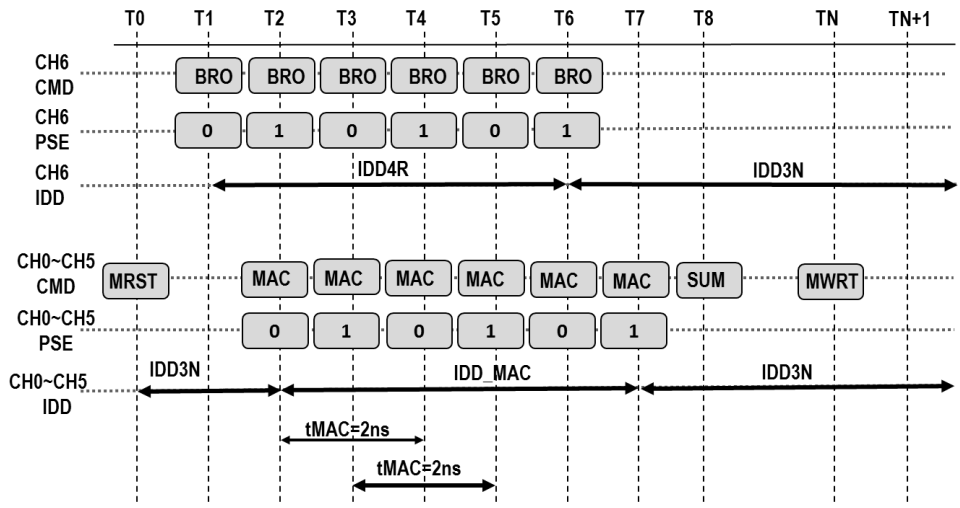


Figure 10: Timing diagram of McDRAM

5. Evaluation Methodology

Neurocube and TETRIS are the papers that operate NN using existing 3D-stacked DRAM. However, these methods also cause performance degradation due to the memory bandwidth because of the limitation of the TSV line in the DRAM. Since this method has 2.5 KB of SRAM per channel, it is necessary to continuously fetch data through the TSV line in order to access memory required for large matrix processing. In Table 2, data access through the TSV line consumes less power than data access through external I/O, but consumes much more power than data access through GIO.

On the other hand, the TPU uses a very large size of 24MB SRAM inside and since it has 64K 8bit MACs, we thought it will achieve higher performance than the previous two papers when handling large Matrix. McDRAM is useful for handling very large-sized matrices such as MLPs and RNNs, rather than processing CNNs workload. So we compared McDRAM with TPU in MLPs and RNNs. In addition, we also experimented with the recently released GPU (NVIDIA P4), a very power efficient GPU for inference.

5.1 Workloads

In order to compare with the TPU, we used the workload used in the TPU. However, since the contents of the correct workloads cannot be known, we conducted the experiment by inferring the matrix size as shown in Table 4 below. We calculated the matrix size with the weights size and the total number of layers presented in the TPU paper. For example, MLP0 has a total weight of 20M and a total of 5 FCs, so it is assumed that it has 4M per layer and the matrix size is 2K x 2K = 4M. We have experimented with large Matrix operations such as MLPs and LSTMs, considering that all performance is limited by memory bandwidth.

Table 4: Workloads

	FC	Vector	Batch	Matrix Size
MLP0	5		200	2048 x 2048
MLP1	4		168	1144 x 1144
LSTM0	24	34	64	1500 x 1500
LSTM1	37	19	96	980 x 980

As we mentioned, we did not know the exact workload used in the TPU, so we only performed Matrix operations in McDRAM and P4. However, as mentioned in the TPU paper, the performance of

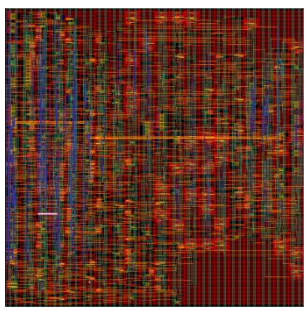
MLPs and LSTMs is limited by memory bandwidth. And effectively, since MLP and LSTM operations are mostly Matrix operations, we compared them with TPU only by Matrix operations. We experimented with P4 using 8bit weight and 8bit activation using CublasGemmEx.

5.2 MAC Unit Simulation

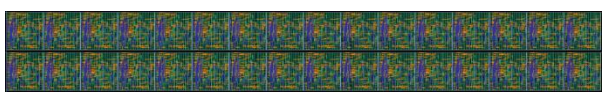
As the speed increases, there are thousands of special registers in the DRAM to compensate for internal signals and to mass-produce memory. These functions exist separately from the DRAM Cell and use a huge number of transistors. DRAM already uses circuits such as Adder logics and Counter logics. However, since complex logic such as MAC has never been used in DRAM, we designed the MAC by using synthesis tool as a method of designing base die of HBM2. We tried to deploy the MAC in the DRAM for the first time, so we occupied enough space and designed it.

Normally, when designing a DRAM, all layouts are fully customized. And unlike logic circuits that operate synchronously with clocks, DRAMs are mostly designed asynchronously with delay adjustments. However, we used the IOSA enable signal as a clock

to design the MAC in synchronous mode to operate in two stages in the 20nm DRAM process. We think that if we design MAC with full custom and if we design asynchronous MAC, we will be able to design less than half the current size.



One MAC Unit (102um x 102um)



16 x 2 MAC per Bank in Pseudo Mode

Figure 11: MAC Layout in 20nm DRAM Scale

Figure 11 shows the layout of a single MAC unit designed with a DRAM 20nm process and 16x2 MACs layout. One MAC has a size of 102um X 102um and operates at the 500MHz clock with 2 stages of pipeline. One MAC operation has a power consumption of about 150uW. When we design DRAM at present, there is no good library for MAC operation, but we will do more research in the future to reduce power consumption and space.

5.3 In-house Simulation

We modified the program to create a behavior modeling program for memory and operate on the new logic in the same way as the existing HBM2. We have modeled a memory controller capable of executing McDRAM commands such as BRO and MAC, and used IDD SPEC and the results of MAC simulation to obtain power consumption. Since the BRO commands used in McDRAM do not actually interface external I/Os, we have modified with IDD to reflect these differences.

6. Evaluation Results

6.1 TOPS/W compared to TPU and P4

McDRAM is currently composed of 96 banks, consisting of a total of 6,144 MACs. And McDRAM is a 4H stack memory, which consists of 4 DRAM cell dies, but it can be operated with more banks if it is composed of 8H stack. Moreover, McDRAM currently only uses one channel for the interface. In a system using existing HBM2, 8 channel interfaces are available, so a total of 8 McDRAMs can be operated in parallel. One McDRAM can perform a maximum of 3.072 TOPS. However, when multiple McDRAMs are installed, the batch can be increased and the TOPS is also increased. For fair comparisons, we have experimented TOPS/W to compare McDRAM with TPU and P4.

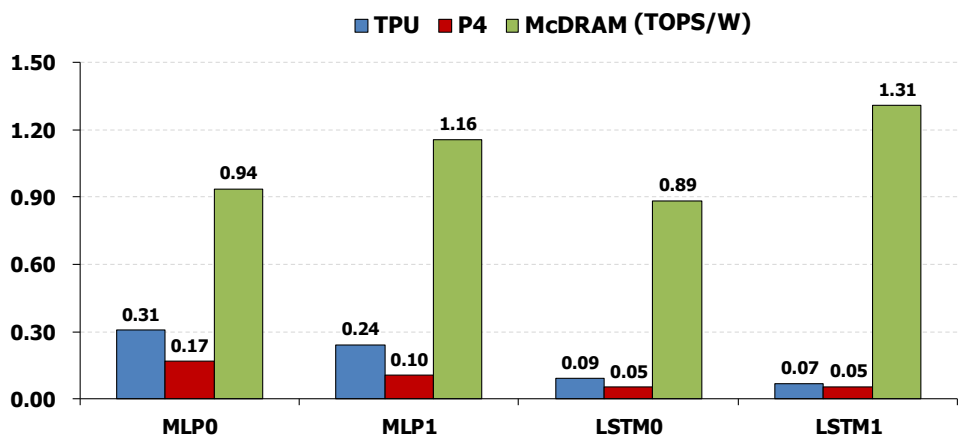


Figure 12: MAC Layout in 20nm DRAM Scale

Figure 12 compares TOPS/W of TPU, P4, and McDRAM. In TPU and P4, TOPS/W gradually becomes weak in the order of MLP0, MLP1, LSTM0, and LSTM1. In the case of LSTM1, as shown in Table 4, there are 37 FCs, so a lot of memory access is required. In case of TPU and P4, we can see that the performance is decreased in case of a lot of memory access, and the memory bandwidth limits the performance. However, McDRAM is affected by the batch size without being affected by the amount of memory access. Since McDRAM only operates with 96 batches, performance cannot be fully achieved if it is not a multiple of 96. Therefore, McDRAM shows better performance in workload using batch 96 as LSTM1.

6.2 Performance Relative to Matrix Size

In general, when the external memory access occurs due to the memory bandwidth constraint, the performance of the Matrix is degraded because the interface is relatively slow compared to the SRAM. McDRAM paradoxically prevents performance degradation due to memory by mounting a MAC unit in memory. If the data can not be reused and the memory needs to be accessed continuously,

McDRAM can access the required memory only once, thus preventing performance degradation due to memory.

We experimented with McDRAM as shown in Figure 13 to see how performance varies with the size of the Matrix. The default matrix size is batch = 96 and the X-size and Y-size are the same 1024 x 1024. We experimented with changing the batch to a multiple of 96, and X-size and Y-size were also performed in multiples of 1024. We confirmed that when the batch was changed, the runtime increased proportionally with the batch size. McDRAM also obtained a proportional increase in runtime when increasing Y Size. Y size is related to the broadcast weight data. Since this data is stored in 16 banks, it can be stored in a very large size and there is no restriction by Y-size due to bank interleaving.

However, when the size of the X-size is more than 8 times, that means 8Kbytes, the runtime increases very much. The X-size is related to the size of the batch. This data is stored differently on the 96 banks. If this data size is less than 8Kbytes, all the data can be fetched by one activation command. Thus, McDRAM can fetch batch data without any Precharge command until all Matrix operations are finished. However, if the batch data size exceeds

8Kbytes, several activation commands are needed. Batch data of 8Kbytes to 16Kbytes requires two activation commands. This is because one page can store only 8Kbytes of data each.

McDRAM must activate all 16 banks to activate the page because the batch data is stored in 16 banks in each channel. As described in Section 4.3, DRAM cannot accept a large number of Active commands at the same time due to current consumption problems. If the memory controller activates one bank, it can activate another bank after tRRD time in the DRAM Specification. In the case of tRRD, tRRD_L and tRRD_S are classified according to the DRAM bank group. tRRD_S is applied when a bank of different bank groups is active, and tRRD_L is applied when a bank of the same bank groups is active. tRRD_L means long tRRD, tRRD_S means short tRRD and tRRD_S is usually one or two clock shorter than tRRD_L. McDRAM has shortened the activation time by activating Active by interleaving among different bank groups, but it still takes a lot of time to activate all.

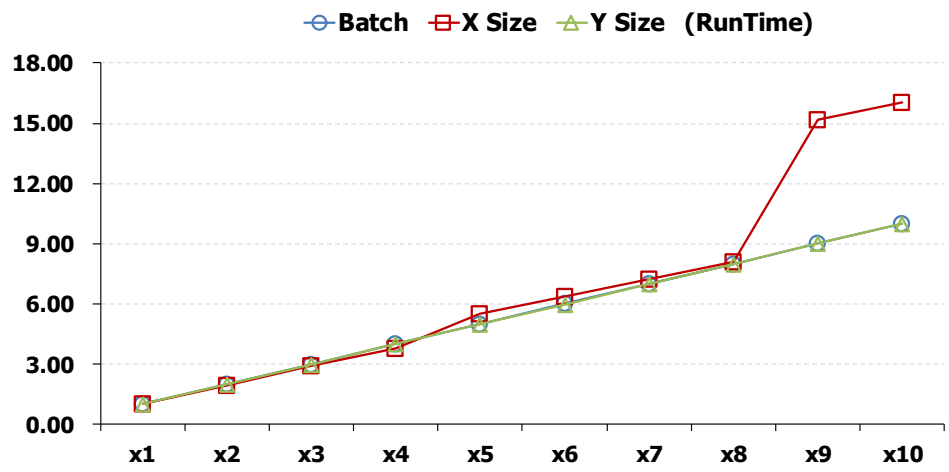


Figure 13: Performance Relative to Matrix Size

6.3 Performance in CNNs, MLPs and LSTMs

We tested ALEXNET, RESNET-152, and VGGNET-16 with McDRAM. Workloads that perform large matrix operations such as MLPs and RNNs broadcast weight data for reuse, but we experimented with CNNs as a method of broadcasting activation data. In CNNs, the result of each layer is stored for each channel of the network. In the next layer, the stored value is broadcast again to perform the convolution operation. Since McDRAM broadcasts activation data, it needs to broadcast the data redundantly, so it exploits data inefficiently compared with large matrix operation such like MLPs and RNNs.

However, since the activation data can be stored and reused

if the size of the buffer is equal to the size of the convolution in each DRAM bank, the power consumption broadcasted through the TSV line can be reduced. For example, in the case of convolution of 8 bits data having a size of 3x3x256, activation data can be reused if the buffer has a total size of 3x3x256 on one bank. This means that a buffer of 2,304 KB is required for one DRAM die. Because activation data is broadcast, it can be transmitted together in 16 banks. However, since the data cannot be used more effectively than the large Matrix, the TOPS when processing CNNs is less effective than MLPs and LSTMs as shown in Figure 14.

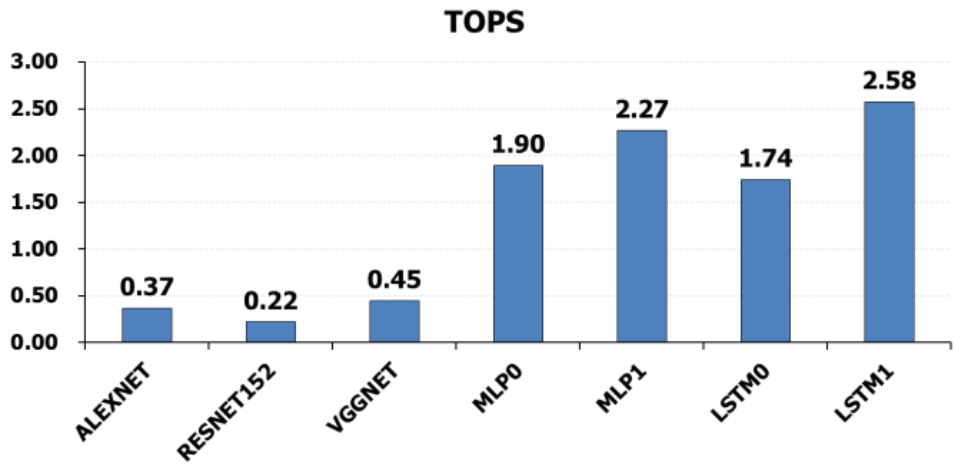


Figure 14: Performance in CNNs, MLPs and LSTMs

McDRAM has the maximum 3.072 TOPS. As shown in Figure 14, McDRAM has the best performance in LSTM1 workload. This means that McDRAM uses memory bandwidth as much as

possible, at 84% of maximum performance. The TPU was only able to use 3% of its peak TOPS in the LSTM1 workload and had the lowest performance, while McDRAM showed the best performance in this workload. We can predict that if we use a workload of deeper layer than LSTM1 and the size of the matrix is larger, the TPU will perform far less than the workload of LSTM1. However, under these conditions, McDRAM can handle the workload while maintaining the same TOPS.

6.4 Energy Breakdown in NNs

We performed energy breakdown on each NNs to analyze the energy used in McDRAM. Figure 15 breaks down energy in all eight channels in McDRAM. Static refers to the energy generated when the conventional DRAM is activated and precharged. Result Write means the energy consumed to combine the MAC result with the Adder operation and prepare it for the next layer. Broadcast is the energy consumed when broadcasting data. Matrix operations are energy consumed when broadcasting weight, and CNN is energy consumed when broadcasting activation data. IOSA is the energy consumed to fetch data from the DRAM cell to the column decoder.

The MAC is the energy consumed when operating the entire MACs.

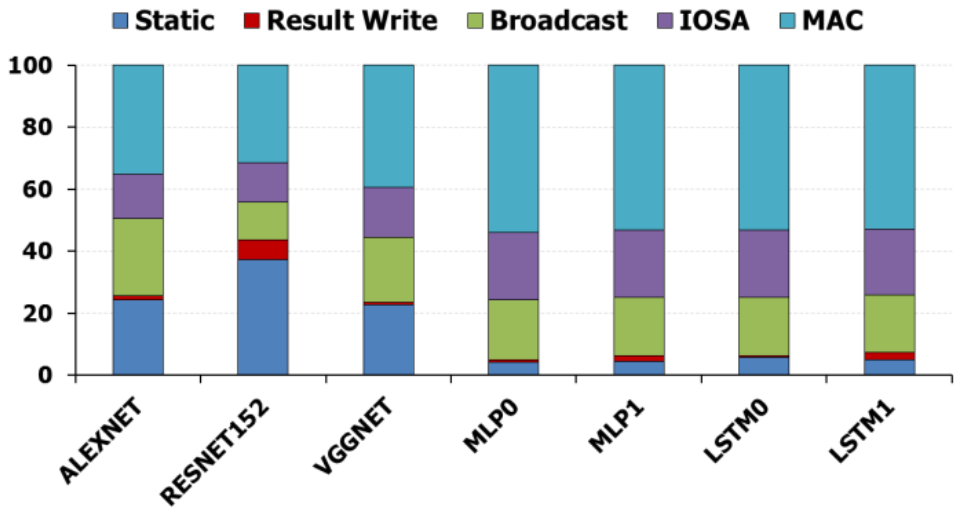


Figure 15: Energy Breakdown of McDRAM

McDRAM does not use the most energy consuming external interface in DRAM. And McDRAM minimizes data movement by placing the MAC circuit nearest to the data that requires MAC operation, so it can consume a lot of energy, which is the most important operation of the MAC. In MLPs and LSTMs, more than half of the total energy is consumed during MAC operation. On the other hand, CNNs consume about 40% of their energy in the MAC. The CNNs result is that the DRAM die does not have an activation broadcast buffer. If we insert buffers, we think we can reduce the energy consumed by broadcasts.

6.5 Overhead

The size of DRAM die in HBM2 is 12 x 8 〖mm〗^2. And currently, the MAC unit size of McDRAM is 102um x 102um. Since a total of 2,048 MACs are contained in one DRAM die, the area occupied by the MAC is 20.48 〖mm〗^2, which causes an overhead of 21.3% in the current DRAM die. However, this experiment was the first attempt to put MAC logics on a DRAM die, so we used a very large layout. We believe that by improving the MAC circuitry and implementing better circuitry, McDRAM can be implemented with much less area.

7. Related Work

To the best of our knowledge, our work is the first to propose a practical hardware architecture that can efficiently execute neural networks inside a standardized, commercial 3D-stacked DRAM. In this section, we briefly discuss previous work relevant to ours, including hardware accelerators for neural networks and processing-in-memory architectures.

Hardware Accelerators. Chen et al. [2] presented DianNao, a hardware accelerator for deep neural networks, and reported orders of magnitude improvement in performance and energy consumption compared with the CPU-based convolution. They also emphasized that off-chip memory accesses dominate the total energy consumption. Chen et al. [6] proposed DaDianNao, which aims at maximizing the capacity of on-chip memory by integrating embedded DRAM on a chip as a large buffer. Du et al. [20] presented a hardware accelerator for embedded vision processing, called ShiDianNao, which reduces on-chip memory accesses by propagating data between processing elements.

Since the efficiency of accelerators depends on (1) making computation efficient and (2) minimizing the on/off-chip memory

accesses, recent accelerator architectures have two common characteristics that address these challenges.

First, several accelerator designs aim at exploiting zero values in weights and/or activations to reduce the amount of computation. Albericio et al. [3] presented Cnvlutin, which groups MACs having the same input activation and tries to skip the broadcast of input activation in case of zero activation. Zhang et al. [7] introduced Cambricon-X, which aims at exploiting zero kernel weights (obtained by weight pruning, e.g., deep compression [21]). It improves the performance by skipping activation transfer and computation associated with zero kernel weights. Chen et al. [8] presented Eyeriss, which applies clock gating on zero input activation for power reduction. Han et al. [32] proposed an efficient inference engine for sparse fully-connected (FC) layer, which utilizes a sparse data format for the weights of pruned FC layer and tries to skip multiplications with zero weights.

Second, some approaches focus on maximizing the data reuse in order to reduce the amount of memory accesses. Eyriss [8] also includes optimizations that (1) broadcast weights and activations to multiple processing elements and (2) compress weights/activations,

both of which contribute to reduction in on-/off-chip memory accesses, respectively. Sim et al. [9] proposed a neural processing engine that aims at reusing kernel weights and input activations by broadcasting them to multiple MACs.

Compared with the aforementioned hardware accelerator designs, our approach is unique in that it enhances the main memory itself with minimal modifications to execute large-scale neural networks inside memory. Through our extensive evaluations, we demonstrated that processing neural networks inside memory has potential of better performance for MLPs and RNNs due to large available memory bandwidth inside of DRAM banks and better energy efficiency for CNNs due to lower energy consumption in memory accesses than hardware accelerators accessing off-chip memory.

Processing-In-Memory (PIM). Recently PIM is being actively studied with the emergence of 3D-stacked DRAM, which can realize cost-effective integration of logic and memory, and the increasing importance of data-intensive applications, which demand very high memory bandwidth. This includes recent research on specialized engines for graph processing [22], MapReduce [23],

data reorganization in memory [24, 25], PIM-enabled instructions [26], HRL [27], transparent offloading and mapping [28], NDA [29], etc.

Along with this trend, there have been several prior works on designing a PIM system for neural networks. Shafiee et al. [30] proposed a PIM architecture for neural networks, called ISAAC, based on a new non-volatile memory technology called ReRAM. They modify the internal structures of an ReRAM crossbar to implement analog-based in-memory computation for neural networks. Chi et al. [31] proposed a similar approach called PRIME, which allows reconfiguration of subarrays into either an array of computation units or memory storage. Our architecture is more practical than such approaches in that ours is based on a widely available memory technology (e.g., HBM2) rather than an emerging memory technology that is not commercialized yet. Also, McDRAM uses digital representations for both numbers and computations, contrary to the previous work based on analog computation, which requires analog-digital data conversion.

In [17], Kim et al. propose an accelerator called NeuroCube which is a two-dimensional array of processing elements on the logic die

of 3D-stacked DRAM. In [10], Gao et al. propose an accelerator called Tetris on the logic die of 3D-stacked DRAM. Compared with the conventional accelerators on the logic chip, NeuroCube and Tetris improve performance and energy consumption by avoiding off-chip memory traffic. Tetris gives further improvements by rebalancing the accelerator architecture, i.e., allocating more silicon area for processing elements thereby increasing the number of processing elements, while decreasing SRAM area, and in-DRAM accumulation.

8. Conclusion

In this paper, we proposed McDRAM, a novel hardware architecture for efficiently executing neural networks inside main memory. McDRAM takes advantage of parallel structures of modern DRAM architectures to exploit abundant parallelism in neural network execution for designing a high-throughput energy-efficient accelerator. Based on the commercial 3D-stacked DRAM architecture (i.e., JEDEC HBM2), our mechanism implements compute units on each DRAM bank and channel selectior on the logic die. For efficient implementation of multiplication operations, we designed three key mechanisms, which are (1) weight broadcast based on the existing I/O bus of DRAM, (2) parallel matrix calculation at 96 of banks for high computation parallelism, and (3) the power efficient output deployment scheme for continuous FC (Fully Connected) operations. McDRAM is controlled by the memory controller, and thus, is flexible to support the execution of various types of neural networks by extending the existing software frameworks for neural networks. According to our experiments based on a commercial in-house McDRAM model and the workload requires the most memory bandwidth (LSTM1),

McDRAM outperforms x18.68 and x23.84 better TOPS/W performance than the state of the art hardware accelerator (Google TPU) and the recent highly efficient GPU (NVIDIA P4), respectively.

References

[1] Google supercharges machine learning tasks with TPU custom chip, https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu.
Accessed: 2017-10-26.

[2] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2014.

[3] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in Proceedings of the International Conference on Computer Architecture (ISCA), 2016.

[4] P. Judd, J. Albericio, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in Proceedings of the International Symposium on Microarchitecture (MICRO), 2016.

[5] To be published. Omitted for blind review.

[6] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A machine-learning supercomputer," in Proceedings of the International Symposium on Microarchitecture (MICRO), 2014.

[7] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-X: An accelerator for sparse neural networks," in Proceedings of the International Symposium on Microarchitecture (MICRO), 2016.

[8] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in International Solid-State Circuits Conference (ISSCC) Technical Digest of Papers, 2016.

[9] J. H. Sim, J. S. Park, M. H. Kim, D. M. Bae, Y. J. Choi, and L. S. Kim, "A 1.42 TOPS/W deep convolutional neural network recognition processor for intelligent IoT systems," in International Solid-State Circuits Conference (ISSCC) Technical Digest of Papers, 2016.

[10] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and efficient neural network acceleration with 3D memory," in

Proceedings of the Architecture Support for Programming Languages and Operating Systems (ASPLOS), 2017.

[11] JEDEC HBM Standard, https://www.jedec.org/news/pressreleases/jedec-updates-groundbreaking-high-bandwidth-memory-hbm-standard/. Accessed: 2017-10-26.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," arXiv preprint arXiv:1512.03385, 2015.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," arXiv preprint arXiv:1603.05027, 2016.

[14] 8Gb B-die DDR4 SDRAM, http://www.samsung.com/semiconductor/global/file/product/2015/08/8G_B_DDR4_Samsung_Spec_Rev1.11_Mar.15-2.pdf. Accessed: 2017-10-26.

[15] 22nm PTM model, http://ptm.asu.edu/. Accessed: 2017-10-26.

[16] NVIDA Deep Learning Platform, http://hpcadvisorycouncil.com/events/2016/china-conference/wp-content/uploads/2016/09/10_NVIDIA-HPC-Advisory.pdf/. Accessed: 2017-10-26.

[17] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory" in Proceedings of the International Symposium on Computer Architecture (ISCA), 2016.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems (NIPS), 2012.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

[20] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting vision processing closer to the sensor," in Proceedings of the International Conference on Computer Architecture (ISCA), 2015.

[21] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in

Proceedings of the International Conference on Learning Representations (ICLR), 2016.

[22] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in Proceedings of the International Conference on Computer Architecture (ISCA), 2015.

[23] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, "NDC: Analyzing the impact of 3D-stacked memory+logic devices on MapReduce workloads" in Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS), 2014.

[24] B. Akin, F. Franchetti, and J. C. Hoe, "Data reorganization in memory using 3D-stacked DRAM" in Proceedings of the International Conference on Computer Architecture (ISCA), 2015.

[25] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "RowClone: Fast and energy-efficient In-DRAM bulk data copy and initialization" in Proceedings of the International Symposium on Microarchitecture (MICRO), 2013.

[26] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," in Proceedings of the International Conference on Computer Architecture (ISCA), 2015.

[27] M. Gao and C. Kozyrakis, "HRL: Efficient and flexible reconfigurable logic for near-data processing," in Proceedings of the International Symposium on High Performance Computer Architecture (HPCA), 2016.

[28] K. Hsieh, E. Ebrahimi, G. Kim, N. Chatterjee, M. O' Connor, N. Vijaykumar, O. Mutlu, and S. Keckler, "Transparent offloading and mapping (TOM): Enabling programmer-transparent near-data processing in GPU systems," in Proceedings of the International Symposium on Computer Architecture (ISCA), 2015.

[29] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim, "NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules," in Proceedings of the International Symposium on High Performance Computer Architecture (HPCA), 2015.

[30] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams and V. Srikumar, "ISAAC: A convolutional neural network

accelerator with in-situ analog arithmetic in crossbars," in Proceedings of the International Conference on Computer Architecture (ISCA), 2016.

[31] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in Proceedings of the International Symposium on Computer Architecture (ISCA), 2016.

[32] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally , "EIE: Efficient inference engine on compressed deep neural network," in Proceedings of the International Symposium on Computer Architecture (ISCA), 2016.

Abstract in Korean

신경망은 대용량 병렬 계산과 높은 메모리 대역폭을 특징으로 합니다. 특히, 메모리 대역폭은 성능을 심각하게 제한하고 전력 소비를 증가시킵니다. 우리는 신경 회로망의 메모리 병목 현상을 극복하기 위해 McDRAM이라는 새로운 메모리 구조를 제안합니다. McDRAM은 DRAM 다이가 내부적으로 신경망을 수행하기 위해 많은 수의 MAC (multiplier-accumulator) 유닛을 갖추고 있습니다. DRAM 메모리의 각 뱅크는 메모리 Pre-fetch 데이터의 크기만큼의 MAC을 여러 개 가지고 있어서 외부 메모리 대역폭보다 훨씬 큰 DRAM의 내부 대역폭을 최대한 활용합니다. McDRAM은 DRAM 데이터 버스를 수정하지 않고 모든 뱅크에 효율적으로 데이터를 Broadcasting 하고 단일 DRAM 명령으로 모든 뱅크에서 MAC 작업을 수행합니다. McDRAM은 최첨단 상업용 메모리 구조 인 HBM2를 기반으로 구현되며 단일 DRAM 패키지에 수천 개의 MAC (HBM2에서 최대 6,144 개)을 장착합니다. 상용 JEDEC HBM2 시뮬레이터를 기반으로 한 내부 메모리 모델을 사용한 우리의 실험에 따르면 McDRAM은 LSTM의 최첨단 하드웨어 가속기 (Google TPU)에 비해 18.68x TOPS / W의 성능을 제공합니다.